# CMR TECHNICAL CAMPUS

# UGC AUTONOMOUS

## Kandlakoya (V), Medchal Road, Hyderabad – 501401

**Accredited by NBA and NAAC with A Grade**

**Approved by AICTE, New Delhi and Affiliated to JNTU, Hyderabad**

## DEPARTMENT OF CSE (AI& ML)


# OPERATING SYSTEM LAB MANUAL

## (R22)COURSE CODE: 22AM307PC

**Prepared by:**

**Mr. V.Ravindernaik**

**Assistant Professor**

1. Write C programs to simulate the following CPU Scheduling algorithms
a)FCFS b) SJF c) Round Robin d) priority.

**a)FCFS**

```
#include<stdio.h>
int main()
{
int bt[20], wt[20], tat[20], i, n;
floatwtavg, tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnterBurstTimeforProcess %d --",i);
scanf("%d",&bt[i]);
}
wt[0] = wtavg = 0;
tat[0]=tatavg=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i] = tat[i-1]+bt[i];
wtavg=wtavg+wt[i];
tatavg=tatavg+tat[i];
}
printf("\tPROCESS \tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=0;i<n;i++)
printf("\n\tP%d\t\t%d\t\t%d\t\t%d",i,bt[i],wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();
}
```

***INPUT***

Enter the number of processes – 3
EnterBurstTimeforProcess0—24
Enter Burst Time for Process 1 – 3
EnterBurstTime forProcess 2 – 3

***OUTPUT***

| PROCESS | BURSTTIME | WAITINGTIME | TURNAROUNDTIME |
|---|---|---|---|
| P0 | 24 | 0 | 24 |

P1 3 24 27
P2 3 27 30
AverageWaiting Time--
17.000000AverageTurnaroundTime--
27.000000

**b)SJF**

```
#include<stdio.h>
intmain()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d",&bt[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0]=tatavg=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg=wtavg+wt[i];
tatavg=tatavg+tat[i];
}
printf("\n\tPROCESS\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverageTurnaround Time --%f",tatavg/n);
getch();
```

}

***INPUT***

Enter the number of processes – 4
EnterBurstTimeforProcess0—6
EnterBurstTimeforProcess1—8
EnterBurstTimeforProcess2—7
EnterBurstTime forProcess3–3

***OUTPUT***

| PROCESS | BURSTTIME | WAITINGTIME | TURNAROUNDTIME |
|---|---|---|---|
| P3 | 3 | 0 | 3 |
| P0 | 6 | 3 | 9 |
| P2 | 7 | 9 | 16 |
| P1 | 8 | 16 | 24 |

Average Waiting Time --
7.000000AverageTurnaroundTime--
13.000000

**c)Round Robin**

```
#include<stdio.h>
int main()
{
int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
floatawt=0,att=0,temp=0;
clrscr();
printf("Enter the no of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnterBurstTimeforprocess%d--",i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnterthesizeoftimeslice--");
scanf("%d",&t);
max=bu[0];
for(i=1;i<n;i++)
if(max<bu[i])
max=bu[i];
for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
if(bu[i]<=t)
{
tat[i]=temp+bu[i];
temp=temp+bu[i];
```

```
bu[i]=0;
}
else
{
bu[i]=bu[i]-t;
temp=temp+t;
}
for(i=0;i<n;i++)
{
wa[i]=tat[i]-ct[i];
att+=tat[i];
awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n);
printf("\nTheAverage Waitingtimeis--%f",awt/n);
printf("\n\tPROCESS\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
getch();
}
```

**INPUT:**

Entertheno ofprocesses– 3
EnterBurstTimeforprocess1–24
Enter Burst Time for process 2 – 3
Enter Burst Time for process 3 – 3
Enter thesizeof timeslice – 3

**OUTPUT:**

| PROCESS | BURSTTIME | WAITINGTIME | TURNAROUNDTIME |
|---|---|---|---|
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

The Average Turnaround time is – 15.666667
TheAverageWaitingtimeis--5.666667

**d)Priority**

```
#include<stdio.h>
int main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
float wtavg, tatavg;
clrscr();
printf("Enter the number of processes --- ");
scanf("%d",&n);
```

```
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter the Burst Time & Priority of Process %d --- ",i);
scanf("%d %d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(pri[i]>pri[k])
{
temp=p[i];
p[i]=p[k];
p[k]=temp;
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;
}
wtavg = wt[0] = 0;
tatavg=tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1] +bt[i-1];
tat[i] = tat[i-1] + bt[i];
wtavg=wtavg+wt[i];
tatavg=tatavg+tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING
TIME\tTURNAROUNDTIME");
for(i=0;i<n;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverageWaiting Time is---%f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
getch();
}
```

***INPUT***

```
Enter the number of processes --5
Enter the Burst Time & Priority of  Process 0---10      3
Enter the Burst Time  & Priority of Process 1 ---1      1
Enter the Burst Time & Priority of  Process2---2        4
Enter the Burst Time &Priority of  Process 3 ---1       5
Enter the Burst Time &  Priority of  Process4---5       2
```

***OUTPUT***

| PROCESS | PRIORITY | BURSTTIME | WAITINGTIME | TURNAROUNDTIME |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 |
| 4 | 2 | 5 | 1 | 6 |
| 0 | 3 | 10 | 6 | 16 |
| 2 | 4 | 2 | 16 | 18 |
| 3 | 5 | 1 | 18 | 19 |

Average Waiting Time is --- 8.200000
AverageTurnaroundTimeis---12.000000

**Viva questions:**

a: What is an Operating system?
Ans: It is an interface between user and Hardware.
b: What is FCFS?
Ans: It is a CPU scheduling algorithm which schedules the processes as first come first serve.
c: What is SJF?
Ans: It is a CPU Scheduling Algorithm which schedules the shortest job first.
d: What is Round Robin ?
Ans: It is a CPU Scheduling Algorithm which works based on fixed time quantum so that each process gets chance to get execute and avoid starvation.
e: What is Priority?
Ans: It is a CPU Scheduling Algorithm which shedules the processes based on the higher priority.

2.Write programs using the I/O system calls of UNIX/LINUX operating system(open,read,write,close, fcntl, seek, stat, opendir, readdir)

**open,read,write, close**

```
#include<sys/types.h>
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>
#define BUFSIZE
512intmain ()
{
int from, to, nr, nw, n;
charbuf[BUFSIZE],ch;
if((from=open("one.txt",O_RDONLY))<0)
{
printf("Error opening source file");
exit(1);
    }

    if ((to=creat("two.txt", O_RDWR)) < 0)


    {
    printf("Error creating destination file");
    exit(2);
    }
while((nr=read(from,buf,sizeof(buf))) !=0)
{
if(nr <0)
{
        printf("Errorreadingsourcefile");
        exit(3);

      }
      nw=0;
      do{
        if((n=write(to,&buf[nw],nr-nw))<0)
        {
        printf("Error writing destination file");
        exit(4);
        }
        nw+=n;
      }
      while (nw <nr);
    }
printf("successfullycopied thecontent fromfiel one.txtto two.txt");

    close(from);
```

```
    close(to);
}
```

**INPUT**
Create file name one.txt with some content OUTPUT
Successfully copied the content from file one.txt to two.txt
//you can see that program has created file two.txt and has content same as one.txt.

**opendir,readdir**

```
#include<stdio.h>
#include<dirent.h>
int main()
{
      Struct dirent* de;
      DIR *dr=opendir(".");
      if(dr==NULL)
      {
            printf("Could not open current Directory");
            return0;
      }
      while((de=readdir(dr))!=NULL)
                  printf("%s\n",de->d_name);
                  closedir(dr);

      return1;
      }
```

**OUTPUT**
// will be list of all the directories;
example output below:ashrc
.bash_history
.bash_logout
.fcfs.c.swp
.landscape
.motd_shown
.msgqs.c.swp
.msgsend.c.swp
.pro.c.swp
.profile
.sc.c.swp
.sudo_as_admin_successful
.viminfo
a.c
a.outd
irdir.c
file1fi
le1.cfi
le2file
2.c

hello.cmq
r.cmqw.c
msgq.txt
msgrecv.c

**stat**

```
#include<stdio.h>
#include<sys/stat.h>
int main()
{
	struct stat sfile;
	stat("stat.c",&sfile);
	printf("st_mode=%o", sfile.st_mode);
	return 0;
}
```

**OUTPUT**
st_mode=100644

**lseek**

```
#include <stdio.h>
#include <unistd.h>
#include<sys/types.h>
#include<fcntl.h>

Void func(char arr[],int n)
{
	int fwrite=open("start.txt",O_RDONLY);

	int fread=open("end.txt",O_WRONLY);

	int count = 0;
	while(read(f_write,arr, 1))
	{
		if(count <n)
		{

		}

		else
		{

lseek (fwrite, n, SEEK_CUR);
```

```
 write(fread,arr,1);
count= n;
count=(2*n);
lseek(f_write,count,SEEK_CUR);
write(f_read,arr, 1);
                        }
             }
             close(fwrite);
             close(fread);
  }

  int main()
  {
  Char arr[100];
  int n;
  n =5;
  func(arr,n);
  return 0;
  }
```

OUTPUT

//Create two file start.txt and end.txt write the content in

one.txt.Youcan seethesleek outputinend.txt.

**fcntl-**creatingawritelock

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include<unistd.h>

int main(int argc,char* argv[])
{
 char* file=argv[1];
 int fd;
 struct flocklock;

 printf("opening%s\n",file);

 fd=open(file,O_WRONLY);
 printf("locking\n");

 memset(&lock,0,sizeof(lock));
 lock.l_type=F_WRLCK;

 fcntl(fd, F_SETLKW,&lock);
```

```
printf("locked;hitEntertounlock...");

getchar ();

printf("unlocking\n");


lock.l_type=F_UNLCK;
fcntl(fd, F_SETLKW,&lock);

close (fd);
return0;
}
```

**OUTPUT**

```
Opening
NULL locking
locked;
hitEnterto unlock...
```

**Viva questions:**

a: what is the use of following functions?
i)open ii) read iii)write iv)close
Ans:
The system call open( ) is used to open a new file
read() is used to read a certain number of bytes starting from the current position in a file
write() is used to write a certain number of bytes into a file starting from the current position
close() is used to close the file

b:what is the use of fcntl() function?
Ans: The fcntl() function provides for control over open files. The fildes argument is a file descriptor.

c:what is the use of lseek() function()?
Ans: It is used to change the position of a file pointer in a file can be done by calling the lseek system call.

d: what is the use of stat() function?
Ans: It is used to read the attributes of a file.

e: what is the use of opendir(), readdir() functions?
Ans: The opendir() function shall open a directory stream corresponding to the directory named by the dirname argument,
The readdir() function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument dirp, and positions the directory stream at the next entry.

2. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
intmain()
{
int alloc[10][10],max[10][10];
intavail[10],work[10],total[10];
inti,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
clrscr();
printf("Enter the no.of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0; for(i=0;i<n;i++)
for(j=0; j<m; j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)work[i]=a
vail[i]; for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A:
for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
```

```
if(c==m)
{
printf("All the resources can be allocated to Process%d",i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
else
printf("\n System is in safe mode");
printf("\nThegivenstateissafestate");
getch();
}
```

**OUTPUT**

```
Entertheno.ofprocessesandresources:43Enter
theclaim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Entertheallocationmatrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:936
All the resources can be allocated to Process 2
Available resources are:62 3
Process2executed?:y
Alltheresourcescanbe allocatedtoProcess3Availableresourcesare:8
3 4
Process3executed?:y
Alltheresourcescanbe allocatedtoProcess4Availableresourcesare:8
3 6
Process4executed?:y
A
lltheresourcescanbeallocatedtoProcess1
```

Availableresources are:936
Process 1 executed?:y
System is in safe mode
The given state is safe state

**Viva questions:**

a:What is a Deadlock?
Ans: Deadlock is a situation where, the execution of two or more processesis blocked because each process holds some resource and waits for other resource held by some other process

b: What are the Necessary conditions for Deadlock?
Ans:There are 4 necessary conditions for deadlock: Mutual Exclusion, Hold and wait, No Preemption, Circular wait.

c:What are the 4 methods of Handling Deadlock?
Ans: There are 4 methods of Handling Deadlock :Deadlock Prevention, Deadlock Avoidance, Deadlock detection and recovery, Deadlock Ignorance

d:What is Bankers Algorithm?
Ans:Bankers algorithm in Operating System is used to avoid deadlock and for resource allocation safely to each process in the system.

e: What is Resource Allocation Graph?
Ans: The resource allocation graph (RAG) is the pictorial representation of the state of a system that gives the complete information about all the processes which are holding some resources or waiting for some other resources.

3. Write a C program to implement the Producer –Consumer problem using semaphores using UNIX/LINUX system calls.

```
#include<stdio.h>
int main()
{
intbuffer[10],bufsize,in,out,produce,consume, choice=0;
in=0;
out = 0;
bufsize=10;
while(choice!=3)
{
printf("\n1. Produce \t 2. Consume \t3. Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case1:if((in+1)%bufsize==out)
printf("\nBufferisFull");
else
{
}
break;;;
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in]=produce;
in=(in+1)%bufsize;
case2:if(in ==out)
printf("\nBufferisEmpty");
else
{
consume=buffer[out];
printf("\nTheconsumedvalueis%d",consume);
out= (out+1)%bufsize;
}
break;
}
}
}
```

***OUTPUT***

1. Produce
2. Consume
3. Exit

Enteryourchoice:2

BufferisEmpty

1. Produce
2. Consume
3. Exit
Enteryourchoice:
1
Enterthevalue:100
1. Produce
2. Consume
3. Exit
Enteryourchoice:2
Theconsumed valueis 100
1. Produce
2. Consume
3. Exit
Enteryourchoice:3

**Viva questions:**

a:What is Critical Section?
Ans: A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action.

b: What are Semaphores?
Ans:Semaphores are integer variables that are used to solve the critical section problem by using twoatomic operations wait and signal.

c: Explain about the types of Semaphores?
Ans:There are two main types of semaphores i.e. counting semaphoresand binary semaphores. Counting Semaphores are integer value semaphores and have an unrestrictedvalue domain. Binary Semaphores are also like counting semaphores but their value is restricted to 0 and 1.

d: What are the 3 classical problems of Synchronization?
Ans:Theclassical problems of process synchronization are: 1. Bounded Buffer (Producer-Consumer) Problem 2. The Readers Writers Problem 3. Dining Philosophers Problem

e: What is a monitor?
Ans:A monitor is essentially a class, in which all data is private, and with the special restriction that only one method within any given monitor object may be active.

5.Write C programs to illustrate the following IPC mechanisms
a)Pipes b)FIFOs c)MessageQueues d)SharedMemory

**Pipes**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
Int main()
{
	int fd1[2];
	intfd2[2];
	charfixed_str[]="Welcome";
	charinput_str[100];
	pid_t p;
	if(pipe(fd1)==-1)
	{
		fprintf(stderr,"pipe failed");
		return1;
	}
	if(pipe(fd2)==-1)
	{
	fprintf(stderr,"pipe failed");
	return1;
	}
	scanf("%s",input_str);
	p=fork();
	if(p<0){
		fprintf(stderr,"forkfailed");
		return1;
	}
	elseif(p>0){
		charconcat_str[100];
		close(fd1[0]);
		write(fd1[1],input_str,strlen(input_str)+1);
		close(fd1[1]);
		wait(NULL);
		close(fd2[1]);
		read(fd2[0],concat_str,100);
		printf("concatenated string %s\n",concat_str);
		close(fd2[0]);
	}
	else{
		close(fd1[1]);
		char concat_str[100];
		read(fd1[0],concat_str,100);
		intk=strlen(concat_str);
```

```
        int i;
        for(i=0;i<strlen(fixed_str);i++)
            concat_str[k++]=fixed_str[i];
            concat_str[k]='\0';
        close(fd1[0]);
        close(fd2[0]);
        write(fd2[1],concat_str,strlen(concat_str)+1);
        close(fd2[1]);
        exit(0);
    }
}
```

**OUTPUT**

```
ubuntu@DESKTOP-B8CV2UR:~$
./a.outHi
concatenated string
HiWelcomeubuntu@DESKTOP-
B8CV2UR:~$
```

**FIFOs**

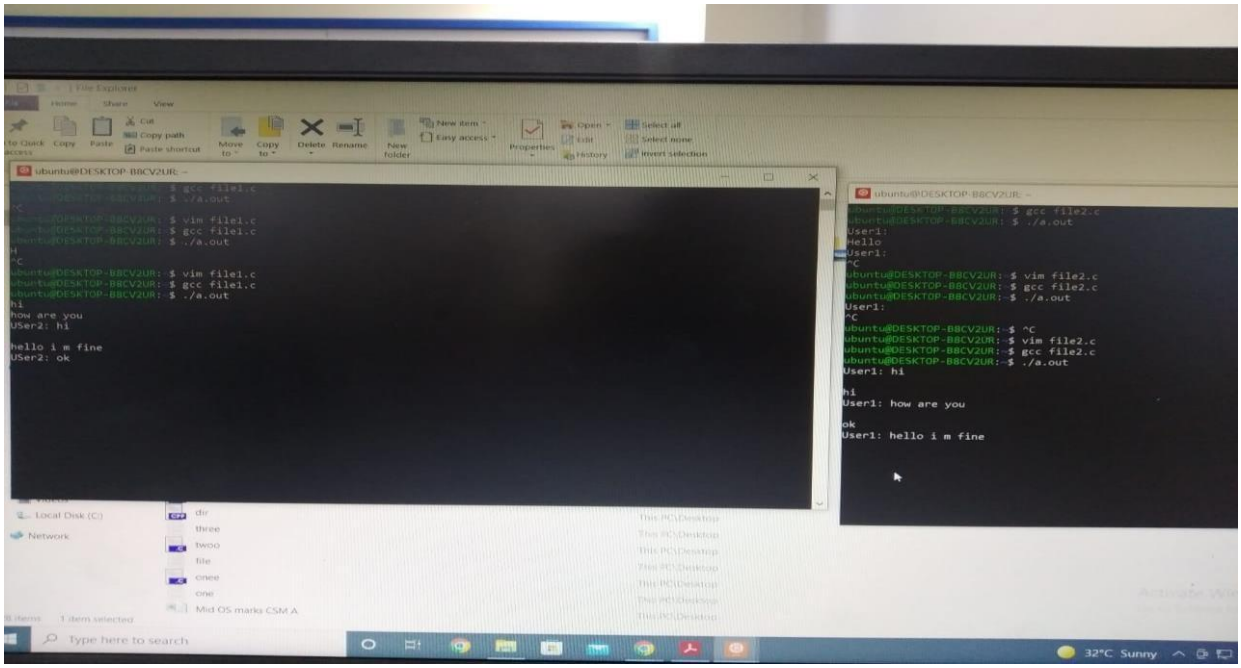
*Fifo writer code*

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
Int main()
{
    int fd;
    char
    *myfifo="/home/ubuntu/myfifo";mkfi
    fo(myfifo,0666);
    chararr1[80],arr2[80];
    while(1)
    {
        fd=open(myfifo,O_WRONLY);
        fgets(arr2,80,stdin);
        write(fd, arr2,strlen(arr2)+1);
        close(fd);
        fd=open(myfifo,O_RDONLY);
        read(fd,arr1,sizeof(arr1));
        printf("USer2: %s\n",arr1);
        close(fd);
    }
    return0;
}
```

*Fiforeaderscode*

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
	int fd;
	char *
	myfifo="/home/ubuntu/myfifo";mkfifo
	(myfifo,0666);
	char str1[80],str2[80];
	while(1)
	{
		fd=open(myfifo,O_RDONLY);
		read(fd,str1,80);
		printf("User1: %s\n",str1);
		close(fd);
		fd=open(myfifo,O_WRONLY);
		fgets(str2,80,stdin);
		write(fd,str2,strlen(str2)+1);
		close(fd);
	}
	return0;
}
```

OUTPUT
//simultaneouslyexecutebothwriteandreadercodeintwoterminals




**MessageQueues**

*//sendercode*

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MAX_TEXT 512
 struct my_msg{
        long int msg_type;
        char some_text[MAX_TEXT];
 };
 int main()
 {
        int running=1;
        int msgid;
        struct my_msgsome_data;
        char buffer[50];
        msgid=msgget((key_t)14534,0666|IPC_CREAT);
        if(msgid ==-1)
        {
                printf("Errorincreatingqueue\n");e
                xit(0);
```

```
        }

        while(running)
        {
                printf("Enter some text:\n");
                fgets(buffer,50,stdin);
                some_data.msg_type=1;
                strcpy(some_data.some_text,buffer);
                if(msgsnd(msgid,(void*)&some_data, MAX_TEXT,0)==-1)
                {
                        printf("Msgnotsent\n");
                }
                if(strncmp(buffer,"end",3)==0)
                {
                        running=0;
                }
        }
}
//Receivercode
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
struct my_msg{
        long int msg_type;
        char some_text[BUFSIZ];
 };
 int main()
 {
        int running=1;
        int msgid;
        struct my_msgsome_data;
        long int msg_to_rec=0;
        msgid=msgget((key_t)12345,0666|IPC_CREAT);
        while(running)
        {
                msg rcv(msgid,(void
                *)&some_data,BUFSIZ,msg_to_rec,0);
                printf("Data received: %s\n",some_data.some_text);
                if(strncmp(some_data.some_text,"end",3)==0)
                {
```

```
                running=0;
            }
        }
        msgctl(msgid,IPC_RMID,0);
}
```

**OUTPUT**

//ExecutionissameasasFIFO

**SharedMemory**

*//Writer*

*code*#include<stdlib.
h>#include<string.h
>#include<sys/shm.h
>#include<stdio.h>#i
nclude<unistd.h>int
main()
{
int i;
void *
shared_memory;charbu
ff[100];
intshmid;shmid=shmget((key_t)2345,1024,0666|IPC_
CREAT);printf("key of shared memory is
%d\n",shmid);shared_memory=shmat(shmid,NULL,0)
;printf("Process attached at
%p\n",shared_memory);printf("Enter some data to
write to shared memory\n");read(0,buff,100);
strcpy(shared_memory,buff);
printf("youwrote:%s\n",(char*)shared_memory);
}

*//Readers Code*

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
Int main(){
    int i;
    void* shared_memory;
    char buff[100];
    int shmid;
    shmid=shmget((key_t)2345,1024,0666);
```

```
    printf("keyof sharedmemory is %d\n",shmid);
    shared_memory=shmat(shmid,NULL,0);
    printf("processattacehedat%p\n",shared_memory);
    printf("Datareadformsharedmemroyis%s\n",(char*)shared_memory);
}
```

**OUTPUT**


**Viva questions:**

a: What is IPC?
Ans: Inter Process Communication (IPC) is a mechanism that involves communication of one process with another process

b: What are Pipes?
Ans: Pipes is used for communication between two related processes, the mechanism used in pipes is half duplex i.e., only one way communication is possible

c: What is FIFO?
Ans: b. FIFO (First-In- First-Out) named pipe are meant for communication between unrelated process, the mechanism used in FIFO is full duplex i.e., same single named pipe can be used for two-way communication

d: What is Message Queue?
Ans: It support communication between two or more processes with full duplex capacity, the processes will communicate with each other by posting a message and retrieving it out of the queue.

e: What is Shared Memory?
Ans: Communication between two or more processes is achieved through a shared piece of memory among all processes, the shared memory needs to be protected from each other by synchronizing access to all the processes.

6. Write C programs to simulate the following memory management techniques
a)Paging b)Segmentation

**Paging**

```
#include<stdio.h>
#include<conio.h>
intmain()
{
 intms,ps,nop,np,rempages,i,j,x,y,pa,offset;
 ints[10], fno[10][20];
printf("\nEnter the memory size -- ");
scanf("%d",&ms);
printf("\nEnterthepagesize--");
scanf("%d",&ps);
nop=ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnternumber of processes--");
 scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
{
printf("\nEnterno.ofpagesrequired forp[%d]--",i);
 scanf("%d",&s[i]);
if(s[i]>rempages)
{
printf("\nMemory is Full");
break;
}
rempages=rempages-s[i];
printf("\nEnterpagetable forp[%d]---",i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np||y>=s[i]||offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{
pa=fno[x][y]*ps+offset;
```

```
printf("\nThePhysicalAddressis--%d",pa);
}
getch();
}
```

**OUTPUT**

```
Enter the memory size – 1000 Enter the page size --
100Theno.ofpages available in memory are--10
Enternumberofprocesses--3
Enterno.ofpagesrequiredforp[1]--
4Enterpagetableforp[1]---8 6
9
5
Enter no. of pages required for p[2]--
5Enter pagetable for p[2] --- 1 4 5 7
3Enterno.ofpagesrequiredforp[3]—5


MemoryisFull
Enter Logical Address to find Physical Address Enter process no. and pagenumber and offset--2
3
60
ThePhysical Address is--760
```

**Segmentation**

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
structlist
{
int seg;
int base;
int limit;
struct list* next;
} *p;
void insert(structlist*q,intbase,intlimit,intseg)
{
if(p==NULL)
{
p=(structlist*)malloc(sizeof(structlist));
p->limit=limit;
p->base=base;
p->seg=seg;
```

```
p->next=NULL;
}
else
{
while(q->next!=NULL)
{
q=q->next;
printf("yes");
}
q->next=(structlist*)malloc(sizeof(structlist));
r- q->next->limit=limit;
q->next ->base=base;
q->next->seg=seg;
q->next->next=NULL;
}
}
intfind(structlist*q,intseg)
{
while(q->seg!=seg)
{
q=q->next;
}
returnq->limit;
}
int search(structlist*q,intseg)
{
while(q->seg!=seg)
{
q=q->next;
}
returnq->base;
}
int main()
{
p=NULL;
int seg,offset,limit,base,c,s,physical;
printf("Entersegment table/n");
printf("Enter -1 as segment value for termination\n");
do
{
printf("Entersegmentnumber");
scanf("%d",&seg);
if(seg!=-1)
{
printf("Enter base value:");
scanf("%d",&base);
```

```
printf("Enter value for limit:");
scanf("%d",&limit);
insert(p,base,limit,seg);
}
}
while(seg!=-1);
printf("Enteroffset:");
scanf("%d",&offset);
printf("Enterbsegmentationnumber:");
scanf("%d",&seg);
c=find(p,seg);
s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf("Addressinphysicalmemory%d\n",physical);
}
else
{
printf("error");
}

}
```

**OUTPUT**

```
Entersegmenttable
Enter-
1assegmentationvalueforterminationEnterseg
ment number:1
Enter base
value:2000Enter value
for limit:100Enter
segment number:2Enter
base
value:2500Entervaluefor
limit:100
Enter segmentation number:-
1Enteroffset:90
Entersegmentnumber:2
Addressinphysicalmemory2590
```

7. Write C Programs to simulate page replacement policies
   a). FCFS
   b). LRU
   c). Optimal

AIM: To implement page replacement algorithms

ALGORITHM:

FCFS:
Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page at the head of the queue
Step 3: Now the new page is inserted at the tail of the queue

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
clrscr();
printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of frames....");
 scanf("%d",&nof);
printf("Enter number of reference string..\n");
 scanf("%d",&nor);
printf("\n Enter the reference string..");
 for(i=0;i<nor;i++)
 scanf("%d",&ref[i]);
printf("\nThe given reference string:");
 for(i=0;i<nor;i++)
 printf("%4d",ref[i]);
 for(i=1;i<=nof;i++)
 frm[i]=-1;
 printf("\n");
 for(i=0;i<nor;i++)
 {
   flag=0;
printf("\n\t Reference np%d->\t",ref[i]);
   for(j=0;j<nof;j++)
   {
     if(frm[j]==ref[i])
     {
       flag=1;
       break;
     }}
      if(flag==0)
   {
     pf++;
```

```
 victim++;
    victim=victim%nof;
    frm[victim]=ref[i];

    for(j=0;j<nof;j++)
    printf("%4d",frm[j]);
   }
 }
printf("\n\n\t\t No.of pages faults...%d",pf);
getch();
}
```

OUTPUT:

FIFO PAGE REPLACEMENT ALGORITHM

Enter no.of frames....4
Enter number of reference string..
6

Enter the reference string..
5 6 4 1 2 3

The given reference string:
..... 5 6 4 1 2 3

Reference np5->5 -1 -1 -1
Reference np6-> 5 6 -1 -1
Reference np4-> 5 6 4 -1
Reference np1-> 5 6 4 1
Reference np2-> 2 6 4 1
Reference np3-> 2 3 4 1

No.of pages faults...6

Ex. No:(b)  LRU PAGE REPLACEMENT ALGORITHM

AIM:  To implement page replacement algorithm
LRU (Least Recently Used)

LRU (Lease Recently Used)
Here we select the page that has not been used for the longest period of time.

ALGORITHM:

Step 1:  Create a queue to hold all pages in memory

Step 2:  When the page is required replace the page at the head of the queue

Step 3:  Now the new page is inserted at the tail of the queue

Step 4:  Create a stack

Step 5:  When the page fault occurs replace page present at the bottom of the stack

/* LRU Page Replacement Algorithm  */

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();

void main()
{
clrscr();
printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of Frames....");
  scanf("%d",&nof);

printf(" Enter no.of reference string..");
  scanf("%d",&nor);

printf("\n Enter reference string..");
  for(i=0;i<nor;i++)
  scanf("%d",&ref[i]);

printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
printf("\n\t The given reference string:");
printf("\n……..");
```

```
  for(i=0;i<nor;i++)
  printf("%4d",ref[i]);
  for(i=1;i<=nof;i++)
  {

    frm[i]=-1;
    lrucal[i]=0;
  }

  for(i=0;i<10;i++)
   recent[i]=0;
   printf("\n");
   for(i=0;i<nor;i++)
   {
     flag=0;
printf("\n\t Reference NO %d->\t",ref[i]);
     for(j=0;j<nof;j++)
     {



        if(frm[j]==ref[i])
       {
            flag=1;
            break;
       }
     }

     if(flag==0)
     {
       count++;
       if(count<=nof)
       victim++;
       else
       victim=lruvictim();
       pf++;
       frm[victim]=ref[i];
       for(j=0;j<nof;j++)
       printf("%4d",frm[j]);
     }
     recent[ref[i]]=i;
   }
printf("\n\n\t No.of page faults...%d",pf);
getch();
}
int lruvictim()
{
  int i,j,temp1,temp2;
  for(i=0;i<nof;i++)
  {
    temp1=frm[i];
    lrucal[i]=recent[temp1];
  }
  temp2=lrucal[0];
  for(j=1;j<nof;j++)
```

```
{

    if(temp2>lrucal[j])
    temp2=lrucal[j];
  }


  for(i=0;i<nof;i++)
  if(ref[temp2]==frm[i])
  return i;
  return 0;
}
```

OUTPUT:

```
                LRU PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3
 Enter no.of reference string..6

 Enter reference string..
6 5 4 2 3 1


          LRU PAGE REPLACEMENT ALGORITHM
       The given reference string:
        ……. 6   5   4   2   3   1

       Reference NO 6->6  -1  -1
       Reference NO 5->        6   5  -1
       Reference NO 4->        6   5   4
       Reference NO 2->        2   5   4
       Reference NO 3->        2   3   4
       Reference NO 1->        2   3   1

No.of page faults...6
```

Ex. No: (c) OPTIMAL(LFU) PAGE REPLACEMENT ALGORITHM

AIM: To implement page replacement algorithms
Optimal (The page which is not used for longest time)

ALGORITHM:

Optimal algorithm
Here we select the page that will not be used for the longest period of time.

OPTIMAL:
Step 1: Create a array
Step 2: When the page fault occurs replace page that will not be used for the longest period of time

/*OPTIMAL(LFU) page replacement algorithm*/

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n.....");
printf("\nEnter the no.of frames");
  scanf("%d",&nof);
printf("Enter the no.of reference string");
  scanf("%d",&nor);
printf("Enter the reference string");
  for(i=0;i<nor;i++)
    scanf("%d",&ref[i]);
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
  printf("\n.....");
printf("\nThe given string");
  printf("\n.....\n");
  for(i=0;i<nor;i++)
    printf("%4d",ref[i]);
  for(i=0;i<nof;i++)
```

```
  {
     frm[i]=-1;
     optcal[i]=0;
  }
  for(i=0;i<10;i++)
     recent[i]=0;
  printf("\n");

  for(i=0;i<nor;i++)
  {
    flag=0;
printf("\n\tref no %d ->\t",ref[i]);
    for(j=0;j<nof;j++)
    {
          if(frm[j]==ref[i])
          {
            flag=1;
            break;
          }
    }
    if(flag==0)
    {
          count++;
          if(count<=nof)
             victim++;
          else
             victim=optvictim(i);
          pf++;
          frm[victim]=ref[i];
          for(j=0;j<nof;j++)
            printf("%4d",frm[j]);
    }
  }
printf("\n Number of page faults: %d",pf);
getch();
}
int optvictim(int index)
{
  int i,j,temp,notfound;
  for(i=0;i<nof;i++)
  {
    notfound=1;
    for(j=index;j<nor;j++)
          if(frm[i]==ref[j])
          {
             notfound=0;
             optcal[i]=j;
             break;
          }
    if(notfound==1)
            return i;
  }
  temp=optcal[0];
  for(i=1;i<nof;i++)
     if(temp<optcal[i])
```

```
 temp=optcal[i];
   for(i=0;i<nof;i++)

       if(frm[temp]==frm[i])

  return i;
 return 0;
}
```

OUTPUT:

OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3
Enter no.of reference string..6

Enter reference string..
6 5 4 2 3 1

OPTIMAL PAGE REPLACEMENT ALGORITHM
The given reference string:
……. 6 5 4 2 3 1

Reference NO 6->6 -1 -1
Reference NO 5-> 6 5 -1
Reference NO 4-> 6 5 4
Reference NO 2-> 2 5 4
Reference NO 3-> 2 3 4
Reference NO 1-> 2 3 1

No.of page faults...6

**Viva questions:**

a: What is Swapping?
Ans: Swapping in the operating system is a memory management scheme that temporarily swaps out an idle or blocked process from the main memory to secondary memory

b: What are the 3 types of Page Replacement Algorithm?
Ans: The 3 page replacement algorithms are: • FIFO Page Replacement Algorithm • LRU Page Replacement Algorithm • Optimal Page Replacement Algorithm
c: what is FIFO Page replacement Algorithm?
Ans: FIFO page replacement algorithm works on the principle of "First in First out", It replaces the oldest page that has been present in the main memory for the longest time.

d: what is LRU Page replacement Algorithm?
Ans: LRU page replacement algorithm works on the principle of "Least Recently Used", It replaces the page that has not been referred by the CPU for the longest time.
e: what is Optimal Page replacement Algorithm?
Ans:Optimal page replacement algorithm replaces the page that will not be referred by the CPU in future for the longest time.